Universidade do Minho
Escola de Engenharia

Departamento de Informática

Licenciatura em Engenharia Informática

Laboratórios de Informática III

Projeto de Laboratórios de Informática III – 3.ª Fase

Grupo n.º 82

Orlando José da Cunha Palmeira, n.º A97755

Braga, fevereiro de 2022

Índice

1 Introdução 2

2 Solução Técnica 3

2.1 Mecanismo de Interação 3

2.2 Testes funcionais e de Desempenho 4

2.3 Gestão dos Dados 5

3 Resultados e Discussão 6

3.1 Medições nos testes do executável 'tests' 6

3.1.1 Resultados Queries Estatísticas 6

3.1.2 Resultados das Queries Parametrizáveis 7

3.2 Discussão dos Resultados 9

3.2.1 Queries Estatísticas 9

3.2.2 Queries Parametrizáveis 9

3.2.3 Ambiente de Execução 10

4 Observações 10

5 Memória gasta nos catálogos 10

6 Conclusões 11

1 Introdução

Nesta última fase do projeto foi proposto que desenvolvêssemos um mecanismo de interação (uma espécie de interface gráfica) que permita ao utilizador utilizar o programa e consultar os dados de uma forma acessível. Este mecanismo inclui um *menu* inicial que apresenta as diversas opções do programa e, quando uma delas for selecionada, apresenta os seus resultados divididos em páginas nas quais o utilizador pode navegar.

Para além do mecanismo de interação, tivemos que desenvolver testes funcionais e de desempenho que avaliam a execução de cada *querie* quanto à sua performance bem como à sua funcionalidade, isto é, o quão certo é o resultado devolvido pela *querie*.

Finalmente, foi ainda proposto que desenvolvêssemos uma boa política de gestão dos dados de forma que o programa responda às funcionalidades pedidas pelo utilizador em pouco tempo, de modo que ocupe pouca memória e permita que a consulta, processamento e armazenamento dos dados não sejam processos muito pesados.

2 Solução Técnica

2.1 Mecanismo de Interação

Quando o utilizador inicia o programa, é-lhe mostrado o *menu* inicial que contém todas as opções disponíveis que ele pode selecionar e o valor numérico que o mesmo deve inserir para poder executar cada uma delas.

Demonstração:

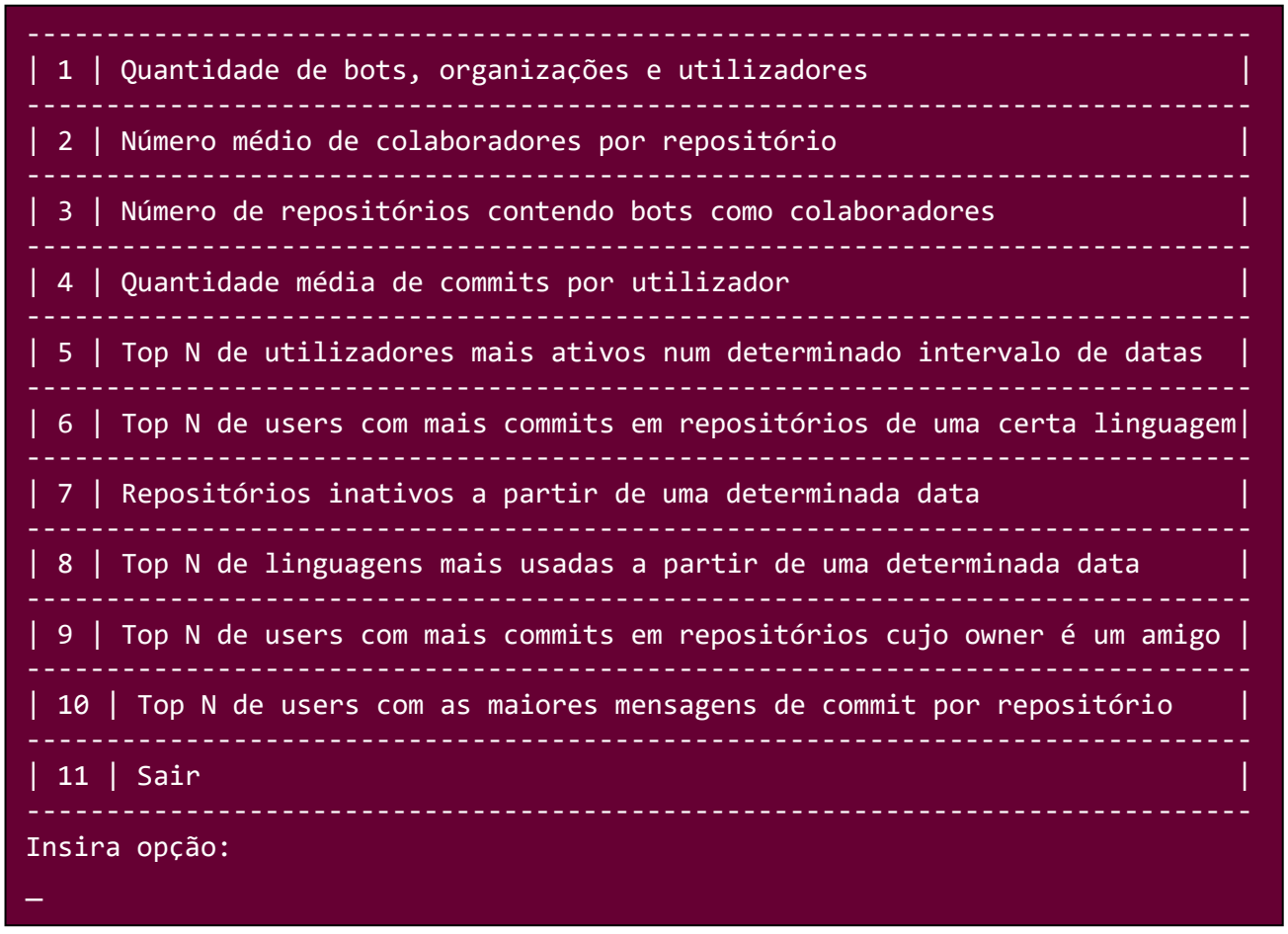

Figura 1 - Menu Inicial

Após a escolha da opção pretendida, o utilizador terá logo acesso à primeira página do resultado.

Cada página dos resultados tem uma altura de 15 caracteres do terminal. É de notar que isto apenas se aplica apenas às *queries* parametrizáveis uma vez que as *queries* estatísticas apresentam apenas valores numéricos que ocupam pouco espaço na janela.

Demonstração de uma possível apresentação dos resultados de uma das queries (página seguinte):

```
--------------------------------------------------------------------------------
ID | LOGIN | QUANTIDADE DE COMMITS
651685|login1|16
651686|login2|21
651687|login3|36
651688|login4|27
651689|login5|98
651690|login6|42
651691|login7|55
651692|login8|125
651693|login9|164
651694|login10|500
651695|login11|30
651696|login12|30
651697|login13|23
651698|login14|96
651699|login15|52
------------------------------ Página 1 de 100 -------------------------------
P        -> Próxima Página
A        -> Página Anterior
S <n>    -> Saltar para a página n
M        -> Voltar ao menu principal
Insira opção:
_
```

Figura 2 - Possível apresentação de resultados de uma querie

Como se pode ver, quando os resultados são apresentados, o utilizador dispõe de quatro opções em que três se destinam a navegar pelos resultados e uma para sair e voltar ao *menu* principal.

O programa previne que o utilizador aceda a páginas com índice inválido, isto é, fora do seu intervalo. No exemplo da Figura 2 acima, se o utilizador estiver na página 100 e tentar visualizar a página a seguir (que não existe), o programa automaticamente retorna à página 1. No caso de o utilizador se encontrar na página 1 e tentar ir para a página anterior, o programa transfere-o para a página 100.

2.2 Testes funcionais e de Desempenho

Para a concretização deste exercício, era necessário ter ficheiros com os resultados esperados das *queries.* Uma vez que temos em mãos dados com uma dimensão considerável, criá-los manualmente seria uma tarefa muito difícil e dispendiosa no que toca ao tempo necessário. Por esse motivo, criei algumas scripts em Python que geram esses ficheiros com os resultados necessários (o código Python das scripts encontra-se na pasta 'scripts' do repositório).

Nos testes, cada querie tem a sua própria função de teste no módulo 'tests'. Essa função executa a querie dez vezes para um dado argumento e regista o tempo que ela demorou a executar e o seu *hit_rate* (o *hit_rate* é o quociente entre o número de linhas que a querie acertou e o número total

de linhas que a querie tem e, portanto, varia entre 0 e 1 indicando que errou completamente ou acertou tudo, respectivamente) nas dez iterações dos testes.

2.3 Gestão dos Dados

Neste guião, as estruturas que mantém os dados não foram alteradas relativamente ao 2.º guião. A única alteração efetuada foi no módulo 'linkedLists´ em que a estrutura que armazenava o resultado do cálculo da *querie* 7 era uma lista ligada que passou a ser uma árvore binária de procura para melhorar a performance desta querie uma vez que a sua execução era muito lenta (atingia a escala dos minutos).

Para o armazenamento dos dados foram utilizadas as seguintes estruturas:

```
struct user{
        unsigned int id;
        char type;
        unsigned int created_at[3],
            followers,
            *followers_list,
            following,
            *following_list,
           biggest_msg_size;
        char *login;
        CatCommits commits;
};
```

```
struct commit{
    unsigned int repo_id,
        author_id,
        committer_id,
        commit_at[3];
};
```

```
struct repo{
    unsigned int id,
        owner_id,
        created_at[3];
    char *language,
         *description;
};
```

Figura 3 - Estruturas de Dados de armazenamento permanente

Cada uma delas armazena um utilizador, um commit e um repositório, respectivamente, e são inseridas em árvores binárias de procura para efeitos de consulta. Estas estruturas são de armazenamento permanente, o que significa que estão em memória durante toda a execução do programa e são desfeitas apenas quando o utilizador decidir sair.

Para o armazenamento temporário dos resultados das queries temos as seguintes estruturas:

```
struct userList{
    int id, value;
    char *login;
    struct userList *next;
};
```

```
struct repoTree{
    unsigned int repo_id;
    int value;
    char *description;
    struct repoTree *left,*right;
};
```

```
struct languageList{
    char *language;
    int value;
    struct languageList *next;
};
```

Figura 4 - Estruturas de dados auxiliares de armazenamento temporário

Cada uma delas armazena dados de um utilizador, repositório e linguagem, respectivamente. O fator *value* em cada uma delas serve para efeito do cálculo dos resultados das queries. Por exemplo, o *value* na estrutura *userList* pode armazenar o número de commits de um certo utilizador.

Como foi referido, estas estruturas fazem o **armazenamento temporário** dos resultados das queries, o que significa que o seu tempo de uso é muito diminuto e que rapidamente são convertidas em ficheiros. Por esse motivo, irei desconsiderar o seu consumo de memória durante a utilização do programa uma vez que essa memória apenas é utilizada durante o cálculo da querie e durante o processo de escrita da estrutura no ficheiro.

3 Resultados e Discussão

3.1 Medições nos testes do executável 'tests'

3.1.1 Resultados Queries Estatísticas

#Execution	Time taken (seconds)	hit_rate
1	0.000096	1
2	0.000226	1
3	0.000071	1
4	0.000054	1
5	0.000151	1
6	0.000051	1
7	0.000048	1
8	0.000047	1
9	0.000049	1
10	0.00005	1
Average	0.0000843	1
Std Dev	5.96286E-05	0

Tabela 1 - Querie 1

#Execution	Time taken (seconds)	hit_rate
1	0.000057	1
2	0.00005	1
3	0.000049	1
4	0.000049	1
5	0.00005	1
6	0.000049	1
7	0.000049	1
8	0.00005	1
9	0.000049	1
10	0.000048	1
Average	0.00005	1
Std Dev	2.53859E-06	0

Tabela 2 - Querie 2

#Execution	Time taken (seconds)	hit_rate
1	0.00005	1
2	0.00005	1
3	0.00005	1
4	0.000048	1
5	0.000133	1
6	0.000051	1
7	0.000049	1
8	0.000048	1
9	0.00005	1
10	0.000048	1
Average	0.0000577	1
Std Dev	2.64787E-05	0

Tabela 3 - Querie 3

#Execution	Time taken (seconds)	hit_rate
1	0.000055	0
2	0.000048	0
3	0.000049	0
4	0.000048	0
5	0.00005	0
6	0.000049	0
7	0.000049	0
8	0.000048	0
9	0.000047	0
10	0.000048	0
Average	0.0000491	0
Std Dev	2.23358E-06	0

Tabela 4 - Querie 4

3.1.2 Resultados das Queries Parametrizáveis

Command: 50 2000-01-01 2022-02-05			Command: 10000 2000-01-01 2022-02-05		
#Execution	Time taken (seconds)	hit_rate	#Execution	Time taken (seconds)	hit_rate
1	0.89472	0	1	30.37504	0.9022
2	0.88459	0	2	29.97761	0.9022
3	0.879936	0	3	30.10643	0.9022
4	0.899867	0	4	30.09702	0.9022
5	0.88444	0	5	30.12008	0.9022
6	0.873706	0	6	30.5014	0.9022
7	0.87724	0	7	30.14463	0.9022
8	0.874586	0	8	30.50705	0.9022
9	0.875408	0	9	30.14736	0.9022
10	0.881693	0	10	30.46456	0.9022
Average	0.8826186	0	Average	30.24412	0.9022
Std Dev	0.0087184	0	Std Dev	0.196451	0

Tabela 5 - Querie 5

Command: 50 2000-01-01 2022-02-05			Command: 10000 2000-01-01 2022-02-05		
#Execution	Time taken (seconds)	hit_rate	#Execution	Time taken (seconds)	hit_rate
1	2.058616	0.16	1	32.55324	0.9869
2	2.052315	0.16	2	32.40221	0.9869
3	2.07314	0.16	3	32.43951	0.9869
4	2.037168	0.16	4	32.24781	0.9869
5	2.04113	0.16	5	32.43269	0.9869
6	2.031249	0.16	6	32.22814	0.9869
7	2.031965	0.16	7	32.19247	0.9869
8	2.020795	0.16	8	32.04231	0.9869
9	2.014686	0.16	9	32.16004	0.9869
10	2.019245	0.16	10	31.94334	0.9869
Average	2.038031	0.16	Average	32.26418	0.9869
Std Dev	0.018704	0	Std Dev	0.191909	0

Tabela 6 - Querie 6

Command:			Command:		
#Execution	Time taken (seconds)	hit_rate	#Execution	Time taken (seconds)	hit_rate
1			1		
2			2		
3			3		
4			4		
5			5		
6			6		
7			7		
8			8		
9			9		
10			10		
Average			Average		
Std Dev			Std Dev		

Tabela 7 - Querie 7

Command: 30 2015-01-01			Command: 500 2016-01-01		
#Execution	Time taken (seconds)	hit_rate	#Execution	Time taken (seconds)	hit_rate
1	0.383827	1	1	0.316498	1
2	0.398716	1	2	0.317198	1
3	0.39412	1	3	0.321309	1
4	0.392522	1	4	0.318524	1
5	0.405885	1	5	0.322457	1
6	0.402564	1	6	0.320302	1
7	0.398722	1	7	0.321132	1
8	0.393755	1	8	0.32135	1
9	0.394123	1	9	0.320406	1
10	0.392476	1	10	0.315975	1
Average	0.395671	1	Average	0.319515	1
Std Dev	0.00614	0	Std Dev	0.002291	0

Tabela 8 - Querie 8

Command: 100			Command: 10000		
#Execution	Time taken (seconds)	hit_rate	#Execution	Time taken (seconds)	hit_rate
1	2.098239	1	1	31.38818	0.1609
2	2.051976	1	2	31.98444	0.1609
3	2.041528	1	3	31.95801	0.1609
4	2.025861	1	4	31.98394	0.1609
5	2.029763	1	5	31.43954	0.1609
6	2.027637	1	6	32.04756	0.1609
7	2.120301	1	7	32.23761	0.1609
8	2.121594	1	8	31.97007	0.1609
9	2.011405	1	9	31.45658	0.1609
10	2.00444	1	10	32.02203	0.1609
Average	2.053274	1	Average	31.84879	0.1609
Std Dev	0.043996	0	Std Dev	0.301382	0

Tabela 9 - Querie 9

Command: 150			Command: 1500		
#Execution	Time taken (seconds)	hit_rate	#Execution	Time taken (seconds)	hit_rate
1	0.404563	Unavailable	1	0.980278	Unavailable
2	0.401028	Unavailable	2	1.001305	Unavailable
3	0.420608	Unavailable	3	1.014606	Unavailable
4	0.415376	Unavailable	4	1.041683	Unavailable
5	0.39298	Unavailable	5	1.194529	Unavailable
6	0.3874	Unavailable	6	1.265597	Unavailable
7	0.392822	Unavailable	7	1.532618	Unavailable
8	0.393535	Unavailable	8	1.689173	Unavailable
9	0.392916	Unavailable	9	1.746787	Unavailable
10	0.393223	Unavailable	10	1.985474	Unavailable
Average	0.399445	Unavailable	Average	1.345205	Unavailable
Std Dev	0.010941	Unavailable	Std Dev	0.366144	Unavailable

Tabela 10 - Querie 10

3.2 Discussão dos Resultados

3.2.1 Queries Estatísticas

Os resultados obtidos nos testes destas queries já seriam de esperar, uma vez que os parâmetros de que elas necessitam são obtidos em tempo de *parsing* dos ficheiros e guardados sempre no mesmo lugar. Deste modo, é natural que demorem muito pouco tempo a executar uma vez que efetuam cálculos pouco complexos e que obtenham sempre o mesmo resultado pois recebem sempre os mesmos parâmetros.

3.2.2 Queries Parametrizáveis

A querie 5 demonstrou um péssimo *hit_rate* quando o top N de utilizadores é baixo (50). Não creio que isto seja causado por mau funcionamento da querie mas sim pela dimensão dos dados. De facto, é previsível que os cálculos que o programa faz sejam diferentes dos das scripts uma vez que as scripts efetuam os cálculos segundo a ordem dos ficheiros e o programa segundo a ordem que os utilizadores estão na árvore que os armazena. Como há milhões de utilizadores, é normal que haja muitos deles com um 'empate' no número de commits e, por isso, é possível que alguns possam aparecer nos resultados do programa e não aparecer nos resultados das scripts ou vice-versa.

Algo que eu penso que corrobora esta possibilidade são os resultados da execução da querie 5 com um top N bastante superior (10000). Neste cenário, o *hit_rate* é muito superior e o que pode ajudar a obter estes resultados é o facto de que o horizonte de consulta dos dados é muito superior e os utilizadores que não apareciam quando o top N é 50, passam a aparecer quando o top N é 10000.

O tempo de execução da querie 5 aumentou com o aumento do top N. É normal, uma vez que os resultados da querie são armazenados numa lista ligada e, sempre que lhe é inserido um elemento, é calculado o maior valor dessa lista.

A querie 6 teve uma prestação muito semelhante à querie 5 e penso que os motivos sejam os mesmos que eu referi anteriormente.

Infelizmente, a querie 7 não teve testes pois o tempo de execução da sua função de teste era muito elevado e não me permitiu obter os resultados em tempo útil.

A querie 8 teve uma prestação bastante boa sem grandes alterações do tempo de execução (surpreendentemente o tempo até desceu com o top N maior) e com um *hit_rate* de 100 %. Não é uma querie muito pesada nem muito complexa e penso que seja por isso que tenha resultados tão bons.

A querie 9 teve resultados estranhos quanto ao *hit_rate.* De facto, é difícil perceber como é que com um horizonte de pesquisa maior, isto é, com o aumento do top N, se pode obter uma interseção dos resultados obtidos com os resultados esperados menor. Realmente, não consigo perceber de onde estes resultados possam vir.

Quanto ao tempo de execução da querie 9, creio que seja normal o aumento quando temos um top N maior. Tal como na querie 5, este aumento do tempo de execução pode ser gerado pelo tamanho da lista ligada que armazena o resultado desta querie.

A querie 10, infelizmente, não tem *hit_rate*. Isto deve-se ao facto desta querie não ter conseguido ser concluída com sucesso e, por esse motivo, não há ficheiros com resultados esperados para a mesma.

A querie 10, em termos de execução, é muito semelhante à 5 pois usa a mesma estrutura de dados. Por esse motivo, ela teve um ligeiro aumento no tempo quando se aumenta o top N. Ela é bastante mais rápida

do que muitas das anteriores pois os parâmetros que ela necessita são calculados em tempo de *parsing* dos ficheiros e, por isso, ela só tem de escrever o resultado.

3.2.3 Ambiente de Execução

Asus ZenBook 14 UM425IA-R7ARDSB1
SO: Linux Ubuntu 20.04
CPU: AMD Ryzen 7 4700U Octa-Core, Clock básico: 2.00 GHz, Clock max boost: até 4.1 GHz
Memória RAM: 16GB LPDDR4X on board , 15.4 GB utilizável
Disco: SSD 512 GB

4 Observações

- Todos os resultados referidos neste relatório são relativos do *dataset* de ficheiros fornecidos pela equipa docente para o guião 3.
- O mecanismo de interação com o utilizador está adaptado para o tamanho standard de um terminal, isto é, 80x24 caracteres.
  Também pode ter algumas falhas quando uma linha é demasiado comprida e tenha que utilizar duas para se ver completamente.

5 Memória gasta nos catálogos

Total: 345 575 178 bytes = 329.56617164 MB

Estes valores foram obtidos pela script memory.py na pasta ‘scripts’ do repositório que calcula o valor aproximado de quanto gastam os catálogos de utilizadores, repositórios e commits.

Infelizmente, não consegui utilizar o valgrind uma vez que esse programa tinha uma execução muito lenta.

6 Conclusões

Os resultados obtidos nas queries estão aquém do que eu gostaria de ter na realidade. Gostaria que todas elas conseguissem atingir os tempos de execução abaixo dos 5 segundos referidos no guião e, para além disso, que tivessem tido uma taxa de acerto de, pelo menos, 90%. Algo que não aconteceu em todas elas.

O mais desanimador nesta etapa final é, com certeza, não ter conseguido concluir a querie 10.

Algo que talvez pudesse ter melhorado a performance geral do programa seria a utilização de uma biblioteca *standard* do C como, por exemplo, a GLib. Eu tentei por diversas vezes usar esta biblioteca mas não consegui instalá-la nem configurá-la no meu editor de texto (Visual Studio Code).

Gostaria de ter tido mais tempo para melhorar o mecanismo de interação/interface. Com o uso da biblioteca ncurses teria ficado com um visual mais apelativo e gostaria também de ter tido mais tempo para adaptar o sistema de paginação ao tamanho da tela do terminal para o utilizador não ter de ficar com o terminal no tamanho standard de 80x24 caracteres.

Por fim, algo que também não é bom, é o facto de a querie 7 não ter conseguido ser testada devido à lentidão da sua função de teste gerada pelo tamanho dos ficheiros de output desta querie.